Preview Coordination: an Enhanced Execution Model for Online Scheduling of Mobile Manipulation Tasks

Enea Scioni[1,2], Markus Klotzbuecher[2], Tinne De Laet[2], Herman Bruyninckx[2] and Marcello Bonfè[1]

***Abstract*— Task specification models define the activities to be executed by a robot in order to achieve its goal. Classical examples are the sequences involved in assembly or pick and place tasks. This work introduces the *preview coordination* execution model, an extension to the traditional way in which the execution of such task specifications is coordinated at runtime. Instead of taking activities one-by-one as defined in the task specification model, preview coordination optimizes the task scheduling based on knowledge about the likelihood that not just the activities required by the current state can be executed, but that also one or more of those related to future states of the system can be activated. An experiment with mobile manipulation tasks illustrates the benefits of preview coordination.**

I. INTRODUCTION

Coordination, sometimes called orchestration, is concerned with managing computations so that a given composite system behaves as intended. As a special case, *task (execution) coordination* executes and monitors the individual steps of a task specification such that its goals are reached. A wide range of formalisms have been applied for this purpose in robotics, including Finite State Automata [1], Sequential Function Charts [2], [3], Petri Nets [4] and Statecharts [5], [6]. Complex formalisms like the Task Frame Formalism [7], Stack of Tasks [8], iTaSC [9] or Manipulation Primitives [1] have proven valuable by permitting humans to specify tasks in more intuitive ways. In practice, software frameworks like the ones provided by ROS (actionlib and SMACH [10]) or Orocos [11] (iTaSC and rFSM [12]) are often used as generic software infrastructures for the modeling, execution and monitoring of task specifications.

All of today's task coordination formalisms permit defining sequential and/or parallel execution of activities, and reacting to different sensor processing outcomes. However, to the best of our knowledge, no framework explicitly permits exploiting knowledge about possible *future* activities in order to optimize the overall execution. This work addresses this by introducing *preview coordination execution model* as an extension to the statechart coordination *model* and its *online execution* (or "*scheduling*"). A motivating example is given in Figure 1 which shows a state machine to model the task of grasping an object using a mobile manipulator, as the KUKA youBot shown in Figure 2. This application involves three basic steps: approaching the object, moving the arm to the grasp position and grasping the object. The transitions to *replanning* models the possibility of encountering an obstacle, in which case the path must be re-planned. The key observation is that the first two nominal steps of moving the robot and the arm to the grasp position can (often but not always) be executed simultaneously. That way, the time necessary for positioning the arm before grasping is reduced or even completely eliminated.

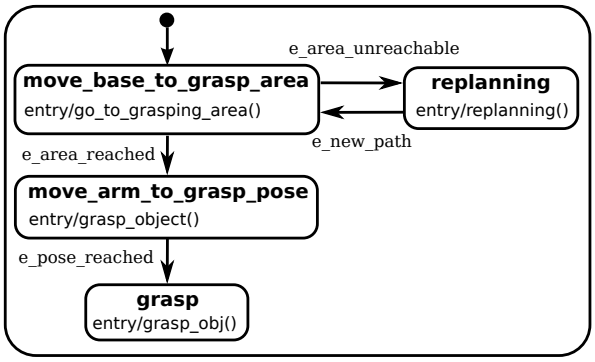

Fig. 1. Simple sequential coordination FSM.

Fig. 2. KUKA youBot mobile manipulator

The paper is structured as follows. Section II is a brief overview on inspiring concepts and related work in different scientific domains. Section III introduces the main concepts behind the *preview coordination* model, and outlines a solving approach using the Finite State Machine (FSM) formalism as coordination model. Section IV presents a case study and results obtained by an implementation subset of the proposed execution model. A short discussion and possible extensions will be provided by Section V.

[1]E. Scioni and M. Bonfè are with the Engineering Department (ENDIF), University of Ferrara, 44122 Ferrara, Italy.

[2]E. Scioni, M. Klotzbuecher, T. De Laet and H. Bruyninckx are with the Department of Mechanical Engineering, University of Leuven, Belgium

Corresponding Author: enea.scioni@unife.it

II. Related Work

The connection between robot *coordination* based on a FSM model and *task specification* formalisms is still a hot research topic. An early attempt to describe assembly tasks, using *manipulation primitive nets*, has been made in [1]. Without directly introducing a task formalism, the combination of the actionlib library and SMACH library [10] offer a FSM based coordination inside the ROS framework [13]. iTaSC [14] is another task specification and execution framework, in which the coordinator role is modeled by so-called *skills* [15], that contain parameter configurations of different motion specifications. A task optimization scheduling based on the *Stack of Tasks* approach [16] has been presented in [17] and [18], however no explicit coordination action is included. Furthermore, resource allocation and task scheduling is a well-known problem in embedded system design and several solutions have been proposed by the Operations Research community. In particular [19] offers a constraint-programming solution for the scheduling of so-called *Conditional Task Graphs* (CTG), directed acyclic graphs whose nodes represent activities, linked by arcs representing precedence relations. Although CTGs have similarities with FSM coordination models, these techniques are not directly applicable in the robotics context, because of the following reasons: (i) in CTGs scheduling, the problem is defined and solved offline, while robotics applications often require runtime adaptations due to uncertain knowledge of the environment; (ii) the CTGs task model is time-based, while a robotic task specification model does not always involve time constraints and, even in that particular case, task duration commonly depends on other physical constraints, such as acceleration and velocity limits.

Scheduling activities is also considered a hot research topic in robotics. Solutions based on "motion primitives" have been investigated in [20]. In addition, commercial software frameworks as ABB RAPID or KUKA Robot Language (KRL) already evaluate several lines of code ahead, in order to interpolate commands and perform the optimal "blending" of subsequent motion primitives. Finally, the concept of *preview* as next state prediction is available in the domain of systems and control. For instance, Model Predictive Control (MPC) [21] relies on system models to predict possible future behaviors of the system, from which the optimal input control signals are selected.

III. PREVIEW COORDINATION

The main idea of *preview coordination* is to exploit the robot platform capabilities, by executing non-conflicting future activities concurrently with the current activity. The requirements needed to achieve the previous are a complete *separation* between the *coordination model* and the *execution model* and a *continuous* monitoring of the activities during execution. An overview of the execution model is illustrated in Figure 3.

The first step is to determine a candidate sequence of likely next states (i.e. *preview state list*), starting from the coordination model and the current state. During this process,

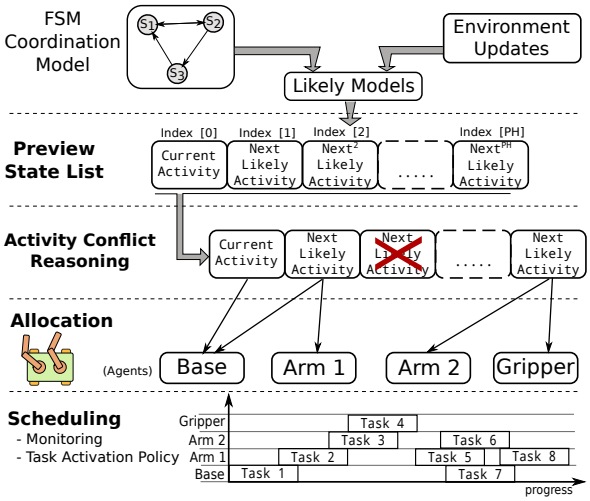

Fig. 3. Preview Coordination Model Overview

environment variability is taken into account through *likely models* (see in Section III-C). Secondly, an *activity conflict reasoning* is applied (Section III-B) to remove conflicting activities described by the *preview state list*. Finally, the activities are scheduled in accordance with the robot platform used and *tasks* are continuously monitored to decide when to execute them (*task activation policy*).

In literature, the terms *task* and *activities* are often interchangeable, sometimes they assume different meanings. The goal of this work is not to introduce a new definition of *task* or a particular *task specification model*. On the contrary, the *preview coordination* execution model is intended to be independent from a specific task model. Thus, the term *task* assumes the meaning of the chosen task specification model and it represents *what* is actually executed at each step. In the trivial case this could be, for instance, a direct command to a trajectory generator for moving the robot to a given pose. Another class of task specification formalisms is *constraint-based*: the command is not embedded in the specification, but only the constraints that have to be satisfied. A typical example is an obstacle avoidance task.

The term *activity* calls the concept of executable entity defined by a set of tasks instances and their configurations. An activity could also provide additional information, such as activity preconditions (conditional constraints to be verified before that the activity is executable) and activity postconditions (constraints related to the end of the activity).

A. Preview State List: a probabilistic approach

Given a FSM coordination model, the first challenge is to determine the most likely *path* of future states, so-called *preview state list*, for which activities may be executed concurrently with the current state activity. The key idea is to associate a probabilistic value to each transition of the FSM model (Figure 4), which represents the likelihood to become active at the next update. In this sense, a FSM coordination model is transformed in a Probabilistic Finite-State Automata (PFA) [22]. Because of the coordination

nature of the FSM model, the current state is known at every step, hence the transformation produces a particular case of PFA, the *Deterministic Probabilistic Finite-State Automata* (DPFA) [22], whose definition is the following:

Definition 1 *DPFA* is a tuple $\mathcal{A} = \langle Q_\mathcal{A}, \sum, \delta_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A}, P_\mathcal{A} \rangle$, where:

- $Q_\mathcal{A}$ is a finite set of states;
- Σ is the alphabet;
- $\delta_\mathcal{A} \subseteq Q_\mathcal{A} \times \Sigma \times Q_\mathcal{A}$ is a set of transitions;
- $I_\mathcal{A} : Q_\mathcal{A} \to \mathbb{R}^+$ (initial-state probabilities);
- $P_\mathcal{A} : \delta_\mathcal{A} \to \mathbb{R}^+$ transition probabilities;
- $F_\mathcal{A} : Q_\mathcal{A} \to \mathbb{R}^+$ final-state probabilities;
- $\sum_{q \in Q_\mathcal{A}} I_\mathcal{A}(q) = 1$;
- $\forall q \in Q_\mathcal{A}, F_\mathcal{A}(q) + \sum_{a \in \Sigma, q' \in Q_\mathcal{A}} P_\mathcal{A}(q, a, q') = 1$;
- $\exists q_0 \in Q$, such that $I(q_0) = 1$ (initial state known);
- $\forall q \in Q, \forall a \in \Sigma, |\{q' : (q, a, q') \in \delta\}| \leqslant 1$.

Recalling from [22], a stochastic language $\mathcal{D}$ is a probability distribution over the set of finite-strings Σ^* (in this context, a string $x \in \Sigma^*$ is a finite and ordered sequence of letters in the alphabet Σ). $Pr_\mathcal{D}(x)$ indicates the probability of a string $x \in \Sigma^*$ under the distribution $\mathcal{D}$. The most probable finite string problem of a DPFA consists of the following:

$$\arg\max_{x \in \Sigma^*} Pr_\mathcal{A}(x). \tag{1}$$

In this probabilistic context, determining the *preview state list* is analogous to the problem reported above. Once that the most probable finite string is known, it is trivial to identify the (likely future) states from the sequence of transitions generated by the string.

A DPFA has the interesting feature that the computational cost of the *parsing* process (i.e. evaluation of $Pr_\mathcal{A}(x)$) depending only on $|x|$ length of the string (in details $\mathcal{O}(|x|)$). Both backward and forward algorithms could be used for the *parsing* process. However, the latter is preferred (Eq. 2), due to the prior knowledge of the current state (as initial state of the *parsing*)

$$Pr_\mathcal{A}(x) = \sum_{q \in Q_\mathcal{A}} \alpha_x(|x|, q) \cdot F_\mathcal{A}(q), \tag{2}$$

where:

$$\begin{aligned} &\alpha_x(0, q) = I_\mathcal{A}(q), \\ &\alpha_x(i, q) = \textstyle\sum_{q' \in Q_\mathcal{A}} \alpha_x(i - q, q') \cdot P_\mathcal{A}(q', x_i, q), 1 \leq i \leq |x|. \end{aligned} \tag{3}$$

Furthermore, the ending condition of the *parsing* process is not guaranteed in general, since the length of the string x is unknown. Introducing the *Preview Horizon* (*PH*) as the number of likely states considered in the *preview* execution and *Maximum Preview Horizon* (*MPH*) as the maximum *PH* allowed (in general $PH \leq MHP$), it is straightforward to see the relationship between $|x|$ and the *MHP*.

To ensure the completion of the parsing process, two alternative solutions can be proposed:

1) Limit on *MPH*: the *MPH* is assumed a priori, fixing the length of the string in the forward algorithm.
2) Encounter a *Steady-State* as ending condition for the forward algorithm. A state $q \in Q_\mathcal{A}$ is defined *Steady-State* if one of the following conditions is true: (i) given a threshold on the final-state probability ($F_T(q)$), if $F(q) \geq F_T(q)$, or (ii) given a threshold on the transition probability ($P_{\mathcal{A}_T}(q, a, q')$), if $\forall q' \in Q_\mathcal{A}$, $\forall a \in \Sigma$: $P_\mathcal{A}(q, a, q') \leq P_{\mathcal{A}_T}(q, a, q')$.

Even though the first solution looks simpler, the second will actually be considered in this work. In fact, setting the *MPH* value a priori is not trivial and it may reduce (*MPH* too small) the activity parallelization of the *preview* execution or, even worse, an expensive and inefficient reasoning phase later (Section III-B). Instead, using *Steady-State*s as ending condition allows a dynamic *MPH* adaptation, without bounding the final result. By tuning the thresholds it is possibile to specify whether the current information is strong enough or not to be used for the *preview* execution. In addition, some states of the FSM are implicitly modeled as *Steady-State*s (see Section III-C).

Obviously, the operations previously described are only possible when all the information on the transitions involved in the parsing are complete. This problem will be treated in Section III-C.

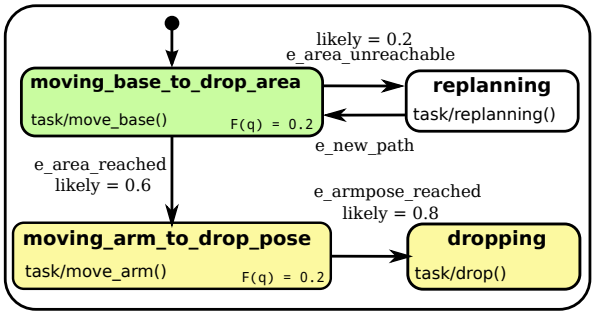

Fig. 4. Snapshot of simple coordination FSM model executed by preview execution model. Green and yellow states indicate, respectively, the current state and the next likely states computed by the preview coordination

For comparison, a similar approach has been used in [19], but since CTGs are acyclic graphs defined a priori, the bounding problem does not occur.

In conclusion, the outcome of this first step is a *preview state list*, in the form of an ordered data structure of activities, starting from the current activity (which will be executed immediately) and terminated by the last predictable future activity obtained by the FSM coordination model analysis. Each of these activities fully describes the tasks to be executed on the robot platform.

B. Activity Conflict Reasoning

The activity conflict reasoning is a mechanism to determine logical conflicts on the activities in the *preview state list*, based on activities precondition and postcondition. These constraints, associated to semantic meanings of the activities, could be expressed symbolically or specified manually. In the first case, a symbolic reasoner implementation is required (e.g. CRAM software toolbox [23]). Searching activity conflicts can be done by iterating the *preview state list* and verifying if the constraints of a target activity are satisfied with respect to all previous activities. If a conflict is detected,

the target activity will not be considered in the allocation and scheduling problem.

A motivating example is illustrated by a dropping operation (Figure 4). Assuming *moving_base_to_drop_area* as current state, the *preview state list* includes also *moving_arm_to_drop_pose* and *dropping* states. It is straightforward to note that the activity described in the state *dropping* (composed of the only task *drop()*) can not be executed in advance because of its preconditions. In fact, since the object has to be dropped in a specific pose, the activity is logically constrained in having the gripper in that pose (precondition). The described reasoning plays a relevant role in large-scale applications. Formalizing the details of this reasoning step is not part of the contribution of this paper, hence it will be considered part of the future work.

C. Determine transition probability values from FSM coordination models (Likely Models)

In Section III-A the discussion has been oriented to propose a method to determine the *preview state list*. However, a solution to compute the probability values online has not been described yet. Before introducing such solution, it is important to define the *meaning* of a probabilistic value associated to transitions in the scope of a chosen FSM coordination model. Without loss of generality, in this work *rFSM* Statecharts [12] (a STATEMATE flavored UML FSM model) are used as coordination model. Since the transition probability is defined as the probability that the transition becomes active, in *rFSM* context the same concept is identified as the possibility that one event is triggering the transition and its guard condition is true.

For the sake of terminology simplification, a transition is called *likely* if the probability associated to the transition is the largest of the probabilities associated to each transition with the same source state. Formally, given a state $q \in Q_{\mathcal{A}}$, a transition (q, a^*, q^*) is *likely* if

$$P_{\mathcal{A}}(q, a^*, q^*) = \underset{a \in \Sigma, q' \in Q_{\mathcal{A}}}{\arg\max} \; P_{\mathcal{A}}(q, a, q'). \tag{4}$$

Furthermore, the term *likely value* refers to the runtime value of the transition probability (i.e. $P_{\mathcal{A}}(q, a^*, q^*)$). Finally, a *likely model* is a computational model responsible to evaluate the *likely value* of an associated transition. Defining a *likely model* is not a trivial task and it has to be analyzed case by case. On the other hand, it is at least possible to define some classes of *likely models*.

The first class of *likely model* is directly related to *prior* information, thus this information is already specified in the FSM coordination model. In most of the cases, this *likely model* is used when a transition is activated by an event (and eventually a guard condition) directly related to the "proper" end of the activity which belong to the source state. In other words, it is marked as *likely* a transition having as target state the next nominal state in the context of the application. A particular case of *prior* information is when a transition is the only one having a particular state as source state. In this case, the transition is likely and the next likely state is immediately identified by its target state. An example is given in Figure 4 by the outgoing transition from *moving_arm_to_drop_pose* state to *dropping* state.

Another interesting class of *likely model* is related to environment changes, thus this information is introduced by *perception* updates. Considering the example in Figure 4, the outgoing transition from *moving_base_to_grasp* to *replanning* state is triggered by the event *e_area_unreachable*, information raised by a *sensor* update. The presence of one obstacle detected along the planned navigation path may increase the probability of transitioning towards the target state *replanning*. According to the definition of DPFA, a direct consequence is that the likely value of the transition going to *moving_arm_to_grasp_pose* decreases. A possible effect is to prevent the execution of a future activity in favor of another one. Furthermore, modern perception algorithms (especially if based on visual features) already provide confidence information related to the main data which are provided. For instance, a simple shape detector could provide number, pose and confidence on the pose of an object identified with a particular shape feature. It is trivial to use such a confidence information to evaluate the likely value of the transition related to that information.

Obliviously, *likely models* based on both *prior* information and *perception* exist. However, *prior likely models* are static, instead *perception likely models* are dynamic and they require an environment update to be evaluated. As a remark, in all the examples previously reported the likely values are mutually exclusive. Mainly this is caused by a *prior* mutual information or due to a *perception likely model* attached to the the same perception feature. However, non-mutual exclusive cases are possible. In such cases, a weighted normalization of the likely values is required.

If the runtime information and the *likely models* do not allow to evaluate a transition as *likely*, its source state becomes a *Steady-State* (by definition, see III-A). The previous guarantee the robustness of the *preview* approach: in the worst case, no future states are selected and the final behavior does not change with respect to the traditional FSM coordination model execution. Finally, some states are implicitly *Steady-State*s because of the activities that they describe. For instance, a robot in stand-by, waiting for a command to start an application is a *Steady-State*. A complete example and further details will be discussed in Section IV.

D. Allocation and Scheduling Problem

This section will show how to determine the task allocation and scheduling from a given *preview state list*.

1) Allocation Problem: a robotic platform is composed of several, heterogeneous *agents*, which interact with each others. Each *agent* is identified by a particular kinematic chain of the robot, and its design is specified to accomplish different operations in the operational space. A serial manipulator could be considered as single *agent*. A gripper mounted on the serial manipulator is another *agent*, that may have itself one or more degrees of freedom. Another example could be a wheeled robotic platform base. In case

of legged robots, each leg can be considered as a separate *agent*. Sensors, especially if equipped with actuators to select the desired Region of Interest (ROI), are *agents* too. Furthermore, two different instances of the same *agent* can co-exist (i.e. dual arm systems). The agents composition fully define the whole kinematic structure of the robot. In other words, a robot is a composite *agent*. The choice to define a robot as a composition of *agents* is not only justified by a kinematic chain definition, but also by a different semantic meaning associated to each *agent*. Practically speaking, an *agent* could be provided by a different vendor, it could implement working modes, its control algorithm may run at a different frequency and it may have an internal behavior. For instance, because of internal reasons (e.g. a fault on an actuator), an agent could be temporarily unavailable.

The allocation problem consists in assigning a task to a proper, available *agent* with respect to the platform capabilities. Therefore, a task specification should include the *type* of *agent* (or a composition of *agents*) involved in its execution (e.g. arm, base, gripper and so on).

A list of tasks to be executed (*task execution list*) is associated to each available *agent*. At runtime, a task is allocated to a specific *agent* by inserting the task description into the *task execution list* of the target *agent*. Once inserted in the list, a task still conserves its activity membership by priority assignment: tasks having membership to the current activities have higher priority, while tasks associated to future activities have decreasing priority. Without loss of generality, we indicate with $i \in \mathbb{N}$ both activity index and its priority. As convention, $i = 0$ for the current activity (higher priority), an increasing value (low priority) for future activities. In case of a simple task specification model (e.g. embedded command), the *task execution list* could be limited to a single element. More advanced task specification models (e.g. constraint-based) allow a larger number of tasks to be inserted in the list. However, a *Task Solver* capable of executing these tasks "as good as possible", exploiting redundancy and solving conflicting constraints according to a priority/weight mechanism is required. Approaches described in the previous sections belong to this task solver category (iTaSC [9], Stack of Tasks [16] and so on).

2) Scheduling Problem: the method described above solves the allocation problem in a deterministic way. However the scheduling problem (deciding *when* execute a given task) is still unsolved. In fact, the allocated tasks belong to different activities (current and futures), and their execution must not affect the current activity tasks. For this purpose, the following parameters are introduced:

- **Activity Progress Parameter (ξ):** for each activity i allocated (i.e. at least one task of the activity allocated), an *activity progress parameter* ($\xi(i) \in [0; 1]$) is defined as dimensionless distance between the activity progress status and the achievement of the activity postconditions. In a nutshell, $\xi(i)$ is a measure on *how far* the i-activity execution is with respect to the ending of the execution. In most cases, ξ is linked to the execution of one "dominant" task described by the activity.
- **Task Execution Cost ($\mathcal{C}$):** A cost related to the execution of the task. This cost could be known a priori (static cost), or it can be computed online through a function specified in the task specification model (e.g. estimated energetic cost to move the end-effector from a starting pose to a goal pose).

Furthermore, we denote by $task(i, j)$ the jth task of the ith activity ($j \in [1; J_i]$, where J_i is the number of tasks described by the ith activity). Therefore, the activation of a future task ($\forall i \geq 1$) can be evaluated from an *utility* function

$$\mathcal{U}_{task(i,j)} = f(\mathcal{C}_{task(i,j)}, \xi(i-1), P_{\mathcal{A}}(q, a^*, q^*)), \qquad (5)$$

trade-off of the arguments indicated:

- **Task Execution Cost** $\mathcal{C}_{task(i,j)}$: a lower cost encourages the execution of the future task. In fact, in case the prediction made by the *preview coordination* is wrong, the cost represents the amount of energy waisted because of the wrong prediction.
- **Progress Parameter of the previous activity** $\xi(i-1)$: in some cases could be useful to activate future tasks only when the ending of the previous (or current) activity is close to be terminated.
- **Likely value** $P_{\mathcal{A}}(q, a^*, q^*)$: the activation depends on *how much* the *preview coordination* is confident about the activation of the transition going to the next likely state.

As remark, the previous does not hold for the tasks with membership to the current activity ($i = 0$), which are always executed by definition. The decision making process around the result of the *utility* function is called *task activation policy*. An example of *task activation policy* is provided by the following:

$$\begin{cases} task(i,j)\ active & \text{if } \mathcal{U}_{task(i,j)} \geq \mathcal{U}_{T\_task(i,j)} \\ task(i,j)\ inactive & otherwise, \end{cases} \qquad (6)$$

where $\mathcal{U}_{T\_task(i,j)}$ is a fixed threshold. In particular, the *task activation policy* above is called *discrete* due to the boolean behavior of the task activation. In conclusion, the scheduling problem has been solved through a continuous *monitoring* of the activities (and the related tasks) already activated in order to decide whenever to activate future tasks.

IV. CASE STUDY

In this section we present a case study (Figure 5) and the results obtained from a basic *preview coordination* implementation. Experiments have been conducted on a KUKA youBot (see Figure 2) mobile manipulator, composed of a holonomic mobile platform and a serial manipulator, with a total of eight degrees of freedom. The software architecture is based on Orocos [11] framework: it includes existing Orocos components to control the youBot, such as standard PID position controllers and trajectory generators to control arm and base. The used trajectory generators create a time constrained path, in accordance with the maximum velocities and accelerations allowed on the youBot.

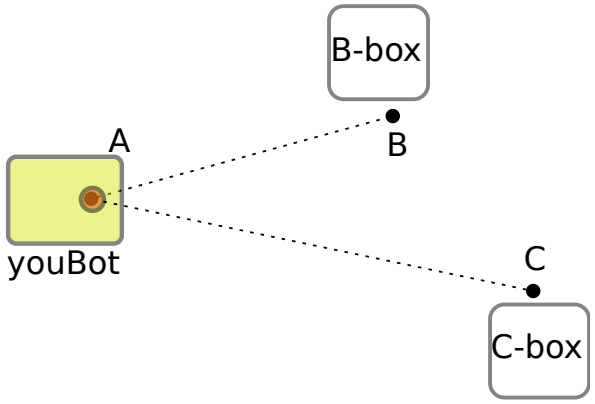

Fig. 5. Experimental Scenario

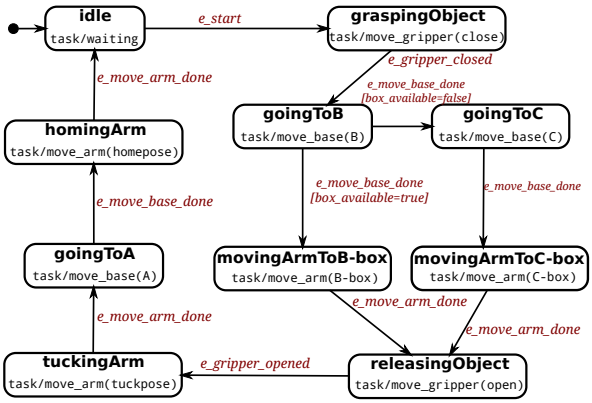

Fig. 6. FSM Coordination Model of the experimental scenario. Transition behaviors (events and guard conditions, in red) are expressed in UML fashion. The latter depends from the FSM coordination model used (rFSM).

A. The Experimental Scenario

Starting from a pose *A*, a mobile manipulation has to release an object (given by a user) into a box placed in location *B* (called *B-box* for simplicity). However, dropping an object into the *B-box* could be not possible due to many factors, among which *box closed*, *box unavailable*, *box full* and so on. If that is the case, the object will be dropped in a box located in C (*C-box*), which is always available. After the dropping, the serial manipulator has to move into a *tuck* pose before navigating back to the original starting pose *A*. Once the pose *A* is reached, the manipulator will be positioned into a homing pose, waiting to receive another object to be dropped. The *B-box* status could be provided either by an external sensor, an on-board sensor (e.g. camera), by the box itself (i.e. a smart box) or a human.

B. The FSM Coordination Model

A possible resulting FSM coordination model (Fig. 6) has been obtained through task decomposition such that every activity contains a single motion task. This unary relationship simplifies the understanding but it does not affect the generality of the approach.

C. The Likely Models

In accordance with the rFSM coordination model, transitions behaviors are modeled through *events* and *guard conditions*. Most of the transitions are triggered by events raised by a continuous monitoring of the ongoing activity. Since each state (apart from *goingToB*) has a single outgoing transition, it is trivial to associate such transitions to a *prior likely model*. The required information is already available in the original FSM coordination model, and the likely value could be a function of the progress parameter ξ. An exception are the out-coming transitions from the state *goingToB*. Both transitions are triggered by the same event, hence the guard condition become "dominant". In this case a *perception likely model* is needed, since the likely value is determined only by a knowledge information (i.e. box available or not). Furthermore, the transitions are mutually exclusive, due to the common environment constraint: no normalization is needed. In conclusion, the likely values are directly associated to the perception information. Notice that the value type depends on the information received: if it is a boolean value, the likely values will switch to its extreme values; if a sensor provides confidence on the information, the likely values will be normalized based on that information.

D. The Allocation and Scheduling problem

On the youBot platform, three *agents* are identified: *Base*, *Arm* and *Gripper*. As a remark, the proposed approach fits completely with the modularity of the youBot. In fact it is possible to support a second serial manipulator, adding the related *agents*. The task specification formalism adopted is simple and a task instance embed directly the command arguments required by the activity. For instance, *move_base(pose)* indicates a task that requires the *Base agent* to be executed, while *pose* is the specific goal sent to the control components. According to Section III-D, the *task execution list* is limited to a single element for each *agent*. During the experiments, the implemented *utility* function is based only on the evaluation of the *progress parameter* ξ of the predecessor activity ($\mathcal{U}_{task(i,j)} = f(\xi(i-1))$. The *task activation policy* used is *discrete* (Eq. 6), hence the resulting behavior is an on/off switching as result of the comparison between the *utility* and a fixed threshold. In this particular case, a future task will be executed after a certain percentage on the previous activity already accomplished.

E. The results

The current implementation is a subset of the whole *preview coordination* execution model. However, these assumptions do not affect the proof of concept. On the contrary, these assumptions point out the effects of a different coordination execution, otherwise not trivial to be verified in a complex scenario and delegating the task execution to an existing task solver. In general, it is not straightforward to evaluate the increasing efficiency of the *preview coordination* execution and a more traditional execution. In fact, the benefits are strongly related to the scenario, its

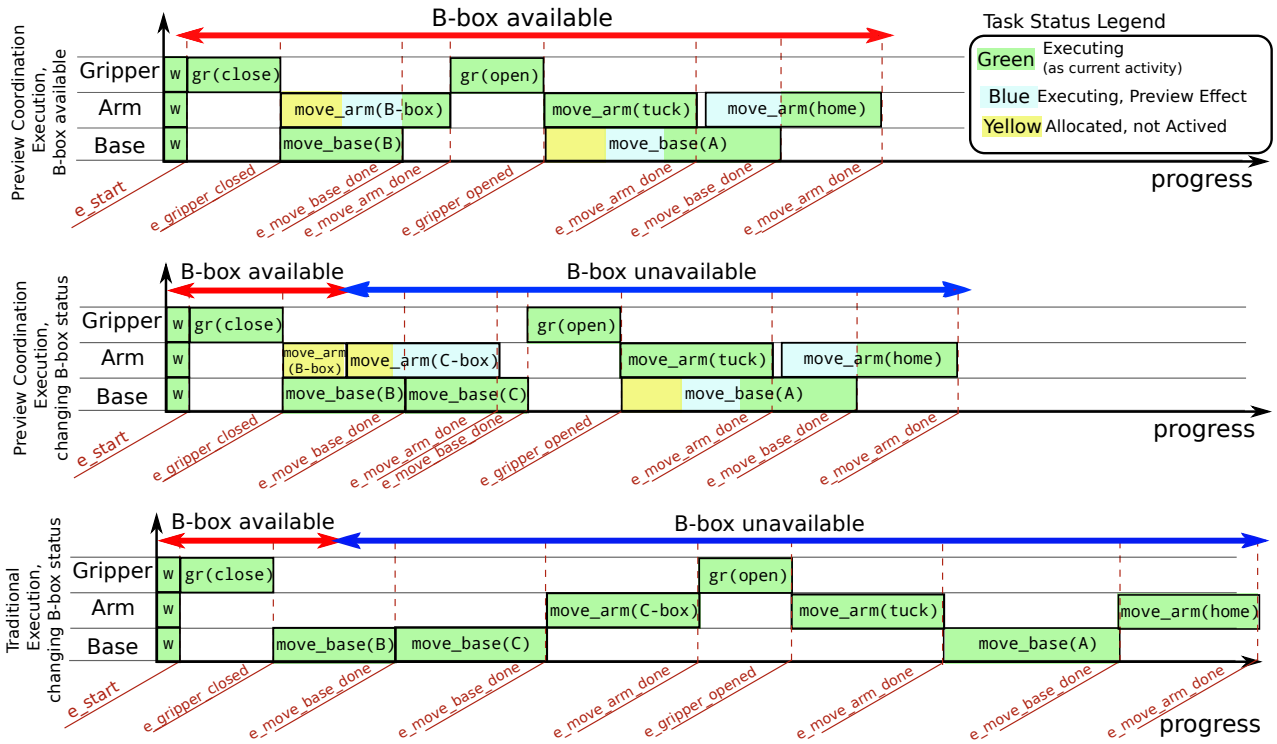

Fig. 7. Different scheduling results from the experimental scenario. *w* indicates the *waiting* task, while *gr* represents the task *move_gripper*. Results are not presented in a time-based fashion, but are normalized as a total execution *progress* (time dependent). From Top to Bottom: (i) *preview coordination execution*, B-box available (best-case). Environment information does not change, hence the task *move_arm(B-box)* has been allocated and then executed in accordance with the *task activation policy*. The *preview* execution effect, as execution of future tasks, is highlighted in *blue*. (ii) *preview coordination execution*, B-box initially available, then unavailable. The task *move_arm(B-box)* has been allocated, then not executed but replaced by the task *move_arm(C-box)* due to environment changes. The mobile platform is still moving to *B* (as current activity), but the *preview* effect allows the arm to be in *C-box* pose in advance. (iii) Traditional execution, without *preview coordination*, case *B-box* not available (worst case). Tasks are executed severally, requiring a larger total execution time with respect to case (ii).

evolution, the task specification used, and the task activation policy implemented. Furthermore, the objective is to increase the whole efficiency of the robot platform, which is not measurable by only one parameter. Obviously, the total time execution seems the most suitable, but it has to be considered as a consequence of the task scheduling parallelization.

As a remark, in the worst case the *preview coordination* is reduced to the traditional execution of the FSM coordination model and the original behavior is preserved.

Figure 7 shown the results obtained from the experimental scenario in different environment conditions, highlighting the effect of the *preview coordination*. The scheduling of future activities speeds up the whole execution, even if an environmental change occurs. Sometimes future activities are terminated while the current activity is still in execution: the time required for the future activities is eliminated and the original desired behavior respected.

Videos of the experiments are available at http://people.mech.kuleuven.be/~s0221775/Videos/Preview/.

V. DISCUSSION AND FUTURE EXTENSIONS

In this section some remarks are briefly reported, illustrating the known limitations of the proposed approach and possible improvements.

1) Complex Scenarios and Likely Models: As previously discussed, it is rather simple to associate a transition to its *likely model* in the proposed scenario. Even if a classification has been proposed, the association has been made manually. An interesting extension is to automate the association process, fetching the proper likely model from an ontology database.

2) Hierarchical FSM coordination models: The proposed *preview coordination* execution model does not support Hierarchical FSM coordination models. However, the ideas introduced in this work are still valid in general and theoretical support for hierarchical models already exists. Hierarchical FSM models will be investigated in the future.

3) Task Solvers and Task Optimization Problem: In the case study, tasks are always defined in the null-space of the other ones, thus parallel tasks in execution are never conflicting. The previous is not a limitation, since the *preview coordination execution* model is not intended to be an alternative solution to task constraint solvers. At the opposite, the *preview coordination* is complementary with existing Task Solvers already mentioned. The role of the proposed execution model is to prepare *online* the optimal problem to be solved, instead to be statically defined by a single activity within a state. Integration with task solvers and performance comparison will be addressed in the future.

4) Scheduling and Task Activation Policy: In Section III-D a task activation policy based on a *utility* function has been introduced. However, in the presented case study only the progress parameter ξ has been considered in the *utility* function. In case of wrong scheduling of a future activity (i.e. its membership state is considered as next likely state, but a perception update remove it from the *preview state list*), a *descheduling* operation could occur (for whose tasks already activated). In general, a task *descheduling* operation is not cost free. Future research will demonstrate that a complex *utility* function will prevent task *descheduling* and its cost. Furthermore, the *task activation policy* used is purely *discrete*: a task could be (fully) activated or not. A *continuous* activation policy, in the sense that the task is executed with reduced configuration (e.g. accelerations and velocities limited), could increase considerably the *preview coordination* effect, reduce the *descheduling* cost, hence improving total efficiency of the robot platform. Finally, a Task Solver having a priority/weight mechanism implicitly supports a *continuous* task activation policy.

VI. CONCLUSIONS

This paper introduces the idea of *preview coordination* as a solution to increase the efficiency of a mobile manipulator. By separation between coordination model and its execution, it becomes possible to execute activities of future states interleaved with those of the current one. During the selection of probable next states, dynamical information, such as environment uncertainty and variability, have been considered. A formalization of the idea has been proposed and a subset of these have been implemented and validated by an experimental scenario. The concepts behind *preview coordination* permit to link cognitive robotics reasoning and task specification formalisms, reinterpreting the role of the coordination as task scheduler formalism. The final result is a performance improvements with respect to the robot platform capabilities. Integration with existing task solvers and extensions will be investigated in the future future work.

ACKNOWLEDGMENT

This research was funded by the European Commission in the FP7 projects BRICS (2008-ICT-231940), RoboHow.Cog (FP7-ICT-288533), SHERPA (FP7-ICT-600958), and by KU Leuven's Concerted Research Action *Global real-time optimal control of autonomous robots and mechatronic systems*. Tinne De Laet is a Postdoctoral Fellow of the Fund for Scientific Research–Flanders (F.W.O.) in Belgium.

REFERENCES

[1] B. Finkemeyer, T. Kröger, and F. M. Wahl, "Executing assembly tasks specified by manipulation primitive nets," *Advanced Robotics*, vol. 19, no. 5, pp. 591–611, 2005.

[2] T. C. . International Electrotechnical Commission, *IEC 61131-3 Ed. 2: Programmable controllers — Part 3: Programming Languages*. IEC, 2003.

[3] A. Hellgren, M. Fabian, and B. Lennartson, "Modular implementation of discrete event systems as sequential function charts applied to an assembly cell," in *Proceedings of the 2001 IEEE International Conference on Control Applications*, 2001, pp. 453–458.

[4] H. Costelha and P. U. Lima, "Robot task plan representation by Petri nets: modelling, identification, analysis and execution." *Journal of Autonomous Robots*, vol. 33, no. 4, pp. 337–360, 2012.

[5] J. C. Marty, A. E. K. Sahraoui, and M. Sartor, "Statecharts to specify the control of automated manufacturing systems," *International Journal of Production Research*, vol. 36, no. 11, pp. 3183–3215, November 1998.

[6] M. Klotzbuecher, R. Smits, H. Bruyninckx, and J. De Schutter, "Reusable Hybrid Force-Velocity controlled Motion Specifications with executable Domain Specific Languages," in *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Francisco, California: IROS2011, 2011, pp. 4684–4689.

[7] H. Bruyninckx and J. De Schutter, "Specification of Force-Controlled Actions in the "Task Frame Formalism": A Survey," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 581–589, 1996.

[8] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A Versatile Generalized Inverted Kinematics Implementation for Collaborative Working Humanoid Robots: The Stack of Tasks," in *Proceedings of the 2009 International Conference on Advanced Robotics*, Munich, Germany, 2009.

[9] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-Based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[10] J. Bohren and S. Cousins, "The SMACH High-Level Executive," *Robotics Automation Magazine, IEEE*, vol. 17, no. 4, pp. 18–20, dec. 2010.

[11] H. Bruyninckx, "Open Robot Control Software: the OROCOS project," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*. Seoul, Korea: ICRA2001, 2001, pp. 2523–2528.

[12] M. Klotzbuecher and H. Bruyninckx, "Coordinating Robotic Tasks and Systems with rFSM Statecharts," *JOSER*, vol. 3, no. 1, pp. 28–56, 2012.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[14] D. Vanthienen, T. De Laet, W. Decré, R. Smits, M. Klotzbuecher, K. Buys, S. Bellens, L. Gherardi, H. Bruyninckx, and J. De Schutter, "iTaSC as a unified framework for task specification, control, and coordination, demonstrated on the PR2," demonstration IEEE/RSJ International Conference on Intelligent Robots and Systems, September 2011.

[15] R. Smits, "Robot skills: design of a constraint-based methodology and software support," Ph.D. dissertation, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium, May 2010.

[16] N. Mansard and F. Chaumette, "Task Sequencing for High-Level Sensor-Based Control," *Robotics, IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 60–72, Feb.

[17] F. Keith, N. Mansard, S. Miossec, and A. Kheddar, "Optimization of tasks warping and scheduling for smooth sequencing of robotic actions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009.*, Oct. 2009, pp. 1609–1614.

[18] F. Keith, P.-B. Wieber, N. Mansard, and A. Kheddar, "Analysis of the discontinuities in prioritized tasks-space control under discreet task scheduling operations." in *IROS*. IEEE, 2011, pp. 3887–3892.

[19] M. Lombardi and M. Milano, "Allocation and scheduling of Conditional Task Graphs," *Artificial Intelligence*, vol. 174, no. 7-8, pp. 500–529, 2010.

[20] M. Vistein, A. Angerer, A. Hoffmann, A. Schierl, and W. Reif, "Instantaneous Switching between Real-Time Commands Continuous Execution of Complex Robotic Tasks," in *IEEE International Conference on Mechatronics and Automation*, 2012, pp. 1329–1334.

[21] V. Duchaine, S. Bouchard, and C. Gosselin, "Computationally Efficient Predictive Robot Control," *Mechatronics, IEEE/ASME Transactions on*, vol. 12, no. 5, pp. 570–578, Oct.

[22] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines—Part I," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1013–1025, 2005.

[23] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM—A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *Proceedings of the 2010 International Conference on Advanced Robotics*, 2010, pp. 1012–1017.